CDVT: a Cluster-based Distributed Video Transcoding Scheme for Mobile Stream Services

Cheng Xu[1], Wei Ren[1,2,3,*], Daxi Tu[3], Linchen Yu[4], Tianqing Zhu[1], and Yi Ren[5]

[1] School of Computer Science, China University of Geosciences, Wuhan, P.R. China
[2] Key Laboratory of Network Assessment Technology, CAS(Institute of Information Engineering, Chinese Academy of Sciences, Beijing, P.R. China 100093)
[3] Guizhou Provincial Key Laboratory of Public Big Data, Guizhou University, Guiyang 550025, China
[4] School of Cyber Science and Engineering, Huazhong University of Science and Technology, Wuhan, P.R China
[5] School of Computing Science, University of East Anglia, Norwich, NR4 7TJ, UK
[*] Corresponding Author: weirencs@cug.edu.cn

Abstract. Distributed video transcoding has been used to huge video data storage overhead and reduce transcoding delay caused by the rapid development of mobile video services. Distributed transcoding can leverage the computing power of clusters for various user requests and diverse video processing demands. However, it imposes a remaining challenge on how to efficiently utilize the computing power of the cluster as well as achieve optimized performance through reasonable system parameters and video processing configurations. In this paper, we design a Cluster-based Distributed Video Transcoding System called CDVT using Hadoop, FFmpeg, and Mkvmerge to achieve on-demand video splitting, on-demand transcoding, and distributed processing, which can be applied to large scale video sharing over mobile devices. In order to further optimize system performance, we conducted extensive experiments on various data sets to find relevant factors that affect transcoding efficiency. We dynamically reconfigure the cluster and evaluate the impacts of different intermediate tasks, splitting strategies, and memory configuration strategies on system performance. Experimental results obtained under various workloads demonstrate that the proposed system can ensure the quality of transcoding tasks while reducing the time cost by up to 50 percent.

Keywords: Distributed Transcoding · Hadoop · Video Processing · FFmpeg.

1 Introduction

Nowadays, mobile devices with media playback capabilities are very common in our daily life [1, 2]. The mobile devices can provide various services [3] and applications, including video and audio[4]. People are getting used to watching videos on their mobile devices, e.g., smartphone, pad, etc. Terminals with diverse Technological specifications and heterogeneous network environment raise new challenges to streaming media services [5]. Various mobile devices needs videos of different qualities, e.g., 240p, 1080p, to meet personalized user requirements [6]. So that video service providers need to prepare video data in multiple bitrates and multiple packaging formats for the same video content on the server-side [7]. In order to provide such services, video transcoding technology was proposed which is now widely used in media services. Traditional video transcoding methods are relatively inefficient and expensive, and it is difficult to support the huge demand for multimedia traffic. Research on new and efficient video transcoding methods has begun to attract attention.

Earlier research relies on multi-node computing power to process transcoding tasks parallelly to reduce transcoding time [8–10]. These methods target special user scenarios, which fail to provide theoretical guidelines and general recommendation paradigms for different user tasks. In the case of a given system, it is desirable to have general recommended configureations which can be friendly adapted to different scenarios. Compared with usual parameter choices, finding options that fit the characteristics of the distributed transcoding method will be more likely to stimulate the system to achieve optimized performance.

In this paper, we propose a Hadoop-based distributed video transcoding system. Our system can transcode a variety of video data according to user's requirements, such as encoding method, video bit rate, and packaging format. In the system, huge volumn user-created video data are stored in the distributed file system of the Hadoop cluster. Then, the video data are processed in a distributed manner, where FFmpeg [11] and Mkvmerge [12] are used for efficient transcoding. Moreover, we conduct extensive experiments to determine the insights and the key points, which have significant impacts on the transcoding efficiency. Our experiment results help us to tune potential factors, process control, system parameter configuration, and segmentation strategies, of the system for optimized system performance. Based on our findings, transcoding algorithms and parameters are well tuned and configured. The experimental results show that the system can obtain notable manifestation by using appropriate segmentation strategy and parameter configuration under the condition of constant cluster size. That is, traditional distributed transcoding increases the speed by 1.35 times compared to single machine transcoding, while our suggested strategies can increase the transcoding rate to 2.06 times.

Based on the aforementioned introduction, the most significant contributions we have made in this paper are as follows:

1) We proposed the architecture of a distributed video transcoding system with optimized performances by evaluating various processing parameters such as cluster size, split size, memory allocation to MapTask.

2) We implemented and extensively evaluated the system CDVT and obtain design parameters and generally recommended transcoding configurations.

3) We designed and initially deployed two key algorithms to enable the user

transcoding configuration to adaptively obtain the optimized transcoding performance.

The rest of this paper is organized as follows: Section 2 surveys related work. In Section 3, we introduces our system. Section 4 proposes the performance analysis. Finally, Section 5 presents conclusions with some future work.

2 Related Work

In recent years, in order to provide users with a good viewing experience and reduce costs for video service providers, distributed video transcoding has become an effective measure to address this challenge [13, 14]. Video transcoding based on distributed clusters can effectively solve the corresponding problems of high hardware and time costs [15]. In this way, computing nodes are added to the distributed computing environment, and the hardware cost of common computing nodes is low, so as to enhance the parallelism of task processing and achieve the purpose of improving transcoding efficiency [16, 17]. Hadoop, as a distributed computing framework proposed by Apache [18], can provide a distributed computing capability by simply deploying it on computer clusters. It has two important frameworks: Hadoop Distributed File System (HDFS) and MapReduce. As a Distributed File System, HDFS has strong scalability and high fault tolerance [19]; MapReduce, as a distributed programming framework, can effectively screen out the underlying details and provide a good environment for developers [20].

For example, Yang *et al.* [21] proposed a Hadoop-based distributed transcoding system that transcodes video files in units of 32MB in size, and concluded that parallel transcoding can reduce transcoding time by about 80%. This study mainly explored the feasibility of a distributed transcoding scheme based on the MapReduce framework, but failed to give more details in terms of system design. Ryu *et al.* [22] designed a Hadoop-based scalable video processing framework to parallelize video processing tasks in a cloud environment. By implementing face tracking experiments, it has been shown that video processing speed can be significantly improved in a distributed computing environment. Moreover, it emphasizes that this video processing framework has good scalability. Kodavalla *et al.* [23] proposesd a distributed transcoding scheme for mobile devices. By using DVC and H.264 advanced video encoders to transcode in the network, the transcoding method has met its key objective of low complexity encoder and the decoder at both mobile devices of video conference application. In addition, researchers such as Sameti *et al.* [17] still want to complete distributed video transcoding on the Spark platform. As a successor of Hadoop, Spark may be found to perform better on distributed transcoding.

While implementing distributed transcoding to reduce video transcoding time, researchers have also begun to explore the factors that affect distributed transcoding. For example, Chen *et al.* [24] proposed a transcoding system based on MapReduce and FFmpeg, which can achieve efficient video transcoding work. On the other hand, this work attempts to introduce a third-party file system in the system, and wants to further improve the transcoding efficiency by reducing the disk I / O and network time overhead during the MapReduce process of the cluster. Song *et al.* [25] also implemented a distributed video transcoding system based on MapReduce and FFmpeg, and experimentally analyzed that the system has different transcoding speed promotion rates for videos of different sizes. It can achieve a speed increase of 1.38 times, 1.51 times, and 1.64 times for 500MB, 1GB, and 2GB video processing respectively. In addition, the effect of the number of cluster nodes on the improvement of transcoding efficiency is demonstrated in a mathematical model. In some transcoding schemes, the video is divided into chunks of equal size and the chunks are distributed across multiple virtual machines for parallel transcoding [26, 27]. However, Zakerinasab *et al.* [28] analyzed the impact of chunk size on coding efficiency and transcoding time. They [29] propose a distributed video transcoding scheme that exploits dependency among GOPs by preparing video chunks of variable size. The scheme effectively reduces bitrate and transcoding time. Kim *et al.* [30] also proposed a Hadoop-based system that can transcode various video formats into the MPEG-4 format. In the experimental part, the experimental scheme is designed for the cluster size, HDFS block size, and backup factor size. Then the impact of the three factors mentioned above on the cluster transcoding speed is evaluated.

In general, the above researches [17, 21, 22, 24, 25, 28–30] have proposed a variety of ideas for the application of processing large-scale video data transcoding, and revealed the advantages of distributed transcoding for traditional transcoding methods when processing large-scale data. On the other hand, they did not further explore the factors affecting the efficiency of distributed transcoding under the proposed mechanism. Therefore, this paper proposes a distributed transcoding system based on Hadoop, and tries to figure out the optimal strategy to adapt the transcoding efficiency of the proposed mechanism while evaluating the system performance through a set of experiments.

3 CDVT: Cluster-based Distributed Video Transcoding System

In this section, we will introduce the overall architecture and workflow of the proposed system in detail, which will include several core components of the system and the functions of each component.

3.1 System Structure Design

Since the system we designed is based on the Hadoop cluster model, the working mode of the cluster nodes is mainly the master/slave mode. When processing video transcoding tasks, the core design of the system consists of placing tasks into the MapReduce framework for processing and generating multiple subtasks for parallel processing to improve the transcoding efficiency of the system.

As shown in Fig. 1, the system is divided into three main domains according to different responsible functions, which are the Video Data Preprocessing and

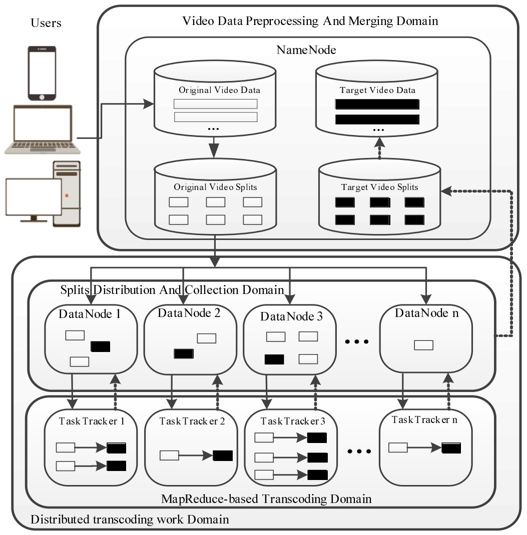

Fig. 1: The overall architecture of the proposed system

Merging Domain (VDPAMD), the Splits Distribution and Collection Domain (SDACD), and the MapReduce-based Transcoding Domain (MTD). The solid and dashed lines in the figure represent the transfer of source data and target data respectively. The main components and functional domains of the proposed system are explained as follow:

NameNode This role is the only global manager in the entire cluster, and its functions include: job management, status monitoring, and task scheduling. More specifically, it is responsible for processing submissions from users in the system, which includes the original video data and XML parameter files that are eager to transform the data. On the other hand, it will start the entire process after accepting user submission as the main program entrance, which includes splitting the original data according to XML content, generating the corresponding index file for each segment, uploading the original segment set to HDFS, invoking the deployed JAR package containing the MapReduce program, downloading the target fragment set to the local, and finally merge the target fragment set and store target data in the specified path.

DataNode This role stores files on HDFS in the form of data blocks in the cluster. Its functions include responding to read and write requests to HDFS and synchronizing data storage information with NameNode. Explaining in more detail, it is responsible for receiving and centrally storing the original and target segment sets in the proposed system. At the same time, when the MapReduce program is invoked, the video fragment and their indexes on the DataNode will be distributed in parallel to the task performers in the cluster. Then, after any task executor finishes transcoding the video fragments, the target fragments will be returned to the DataNode for concentration.

TaskTracker This role is responsible for accepting each subtask job in the cluster, starting and tracking task execution. In other words, after a Map Task is generated, TaskTracker is responsible for executing the task, which includes reading the corresponding splits according to the index file distributed to the task, transcoding the read splits, and writing the transcoded data back to HDFS.

Video Data Preprocessing And Merging Domain The video data preprocessing and merging domain will not only accept video data collection to be transcoded from users such as video surveillance, video service providers, etc., but also collect the transcoded video data collected from the distributed transcoding work domain. Source video splits set and target video splits set, both types of them need to be collected and stored in this domain. When the source data to be transcoded is collected, VDPAMD will process those data according to the parameter submitted by the user. The processing work includes splitting the video according to the configured size, extracting the transcoding parameters from the XML file, and submitting related tasks to the Hadoop cluster. Similarly, when the transcoded splits are collected, the area needs to complete the merging of those splits.

Distributed Transcoding Work Domain The distributed transcoding work domain consists of two parts: the splits distribution and collection domain (SDACD) and the MapReduce-based transcoding domain (MTD).

(1) One between the two, SDACD is implemented based on HDFS, and it will collect the source video splits obtained by VDPAMD and the target video splits completed by MTD. Simultaneously, SDACD also needs to allocate solits for the Map Task on each transcoding node of the MTD according to the index, which is the split distribution function of the domain.
(2) The other between the two, MTD is responsible for executing multiple transcoding subtasks in parallel. Each computing node in the Hadoop cluster will be set up with FFmpeg, then all the nodes with video processing capabilities form MTD. During the execution of the entire MapReduce task, every node of MTD will be assigned one or more Map Tasks, and then the node will find the split that needs to be processed according to the previously generated index and download it to the local for transcoding. Finally, what MTD needs to do is to upload the transcoded splits to SDACD after the Map Task ends.

3.2 System Workflow

After elaborating on the system components and the main functions of each component through the section 3.1, this section hopes to further introduce how they work together through Fig. 2 in order to show the main workflow. Firstly,

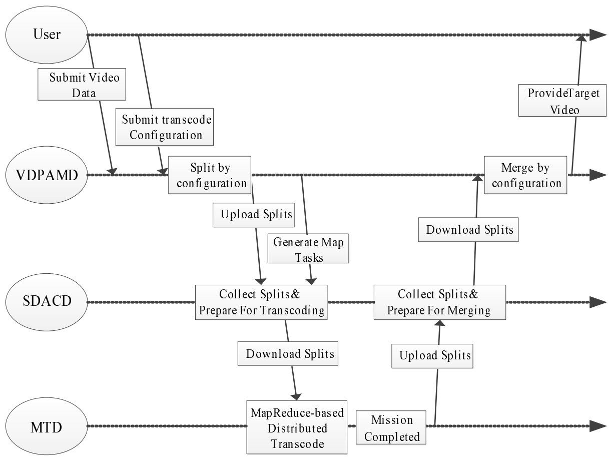

Fig. 2: The flowchart of the transcoding service

users can access the VDPAMD and submit the original video data that needs to be transcoded to this domain. When the video data is submitted, the transcoding options (including bit rate, spatial resolution, temporal resolution, compression standard conversion, packaging format, etc.) selected by the user will be made into the corresponding XML and submitted to VDPAMD together. Secondly, the system processes the video in VDPAMD according to the user's configuration, which includes splitting the original video into multiple fragments and uploading all the fragments to SDACD, and generating a corresponding index for every video fragment. After that, when all the fragments are collected in SDACD, the parameter configuration parsed from the XML submitted by the user will be passed to the MapReduce task and submitted to the Hadoop cluster.

Subsequently, transcoding nodes in the MTD need to transcode the original video fragments. Because of the MapReduce-based working mechanism, it allows every transcoding node to receive one or more Map Tasks. Every Map Task will be assigned an index and transcoding parameters, and they will read the video segment from SDACD according to the data storage information in the index, and then transcode according to the transcoding parameters. Afterwards, when the transcoding task is finished in a distributed computing manner, all nodes in MTD will upload the currently processed segments to SDACD, and then delete all intermediate files. Immediately afterwards, all transcoded shards will be collected on VDPAMD, and they will be cleared in SDACD after they are collected. Finally, VDPAMD merges the fragments completed by the transcoding task, deletes all the fragments after the merge phase is completed, and stores the final target video file to the specified path to meet the needs of the user.

3.3 Key Algorithm Design

Algorithm 1 : Algorithm for selecting optimized splitting size

```
Input:
    $\alpha$: cluster nodes size participating in MTD
    $\beta$: HDFS block size in cluster pre-configuration
    $\kappa$: user-configured split size in XML
    $\omega$: user submitted dataset size
Output:
    $\mu$: optimized split size for current systems
 1: while system available do
 2:     if $\kappa$ != null then
 3:         $\mu \leftarrow \kappa$
 4:         if $\lceil \frac{\omega}{\mu} \rceil < \alpha$ and $\lceil \frac{\omega}{\beta} \rceil < \alpha$ then
 5:             $\mu \leftarrow \lceil \frac{\omega}{\alpha} \rceil$
 6:         end if
 7:         if $\lceil \frac{\omega}{\mu} \rceil < \alpha$ and $\lceil \frac{\omega}{\beta} \rceil \geq \alpha$ then
 8:             $\mu \leftarrow \min\{\mu, \beta\}$
 9:         end if
10:         if $\lceil \frac{\omega}{\mu} \rceil \geq \alpha$ and $\lceil \frac{\omega}{\beta} \rceil \geq \alpha$ then
11:             $\mu \leftarrow \beta$
12:         end if
13:         if $\lceil \frac{\omega}{\mu} \rceil \geq \alpha$ and $\lceil \frac{\omega}{\beta} \rceil < \alpha$ then
14:             $\mu \leftarrow \max\{\lceil \frac{\omega}{\alpha} \rceil, \beta\}$
15:         end if
16:     else
17:         $\mu \leftarrow \beta$
18:     end if
19: end while
20: return $\mu$
```

Regardless of the design parameter level or the transcoding configuration parameter level, the selection that is more in line with the system characteristics can make the CDVT show optimized performance compared to selections that are not suitable. So it should be noted that in terms of video splitting size in VDPAMD and how much computing memory is allocated to each MapTask in MTD, we proposed splitting strategies and memory allocation methods respectively, ensuring CDVT performs as best as possible.

Algorithm 1 provides a pseudo-code to select the optimized split size which was a general recommended configuration obtained through our extensive evaluation of the system CDVT. This method is triggered when the user-defined split size is parsed. The optimized split size for current systems μ are determined by the combined influence of cluster nodes size participating in MTD α, the HDFS block size β in the system configuration, the user-selected partition size κ, and the submitted dataset size ω.

$$\lceil \frac{\omega}{\mu} \rceil < \alpha \; and \; \lceil \frac{\omega}{\beta} \rceil < \alpha \tag{1}$$

When condition (1) is satisfied, the computing nodes in current system cannot be fully utilized no matter whether dataset is split by μ or β , which may be the reason why the dataset submitted by users are too small. Then, with the destination of making the utmost of system resources, the video spliiting size at this case is suitable to be adjusted to $\lceil \omega/\alpha \rceil$.

$$\lceil \frac{\omega}{\mu} \rceil < \alpha \ and \ \lceil \frac{\omega}{\beta} \rceil \geq \alpha \tag{2}$$

When condition (2) is satisfied, it is appropriate for current system that user submitted the dataset at a suitable scale but the system's available computing nodes are underutilized. Correspondingly, it is necessary to properly straighten out the split size so that it approaches β from a larger value.

$$\lceil \frac{\omega}{\mu} \rceil \geq \alpha \ and \ \lceil \frac{\omega}{\beta} \rceil \geq \alpha \tag{3}$$

When condition (3) is satisfied, it is the case that both the scale of user submission tasks and the utilization rate of the computing nodes are appropriate for the current system. At this time, the optimized split size for current systems μ be configured as β preferentially;

$$\lceil \frac{\omega}{\mu} \rceil \geq \alpha \ and \ \lceil \frac{\omega}{\beta} \rceil < \alpha \tag{4}$$

When condition (4) is satisfied, a more extreme case is considered which the size of dataset submitted by the user is small and the split size is also smaller than expected. This situation is also a configuration that is not conducive to the full advantage of current system. The split size needs to be properly corrected in such circumstances so that it approaches β from a smaller value.

Algorithm 2 : Pseudo-Code for checking MapTak Memory Allocation

```
Input:
    $\lambda$: configurable memory size for nodes in MTD
    $\gamma$: memory the user wants to allocate for each MapTask in XML
Output:
    $\epsilon$: memory allocated to MapTask optimized for the current system
1:  while system available do
2:      if $\gamma != null$ and $\gamma \leq \lambda$ then
3:          $\epsilon \leftarrow \gamma$ parse and validate user configuration parameters
4:          if $\lambda \mod \epsilon != 0$ then
5:              resize $\epsilon$ to take full advantage of configurable memory
6:              $p \leftarrow \lceil \frac{\lambda}{\gamma} \rceil$ maximum parallel MapTask numbers under $\gamma$
7:              $\epsilon \leftarrow \frac{\lambda}{p}$ calculate final recommended allocation
8:          end if
9:          $\epsilon \leftarrow \lambda$
10:     end if
11: end while
```

Algorithm 2 provides the pseudo-code for checking MapTask allocated Memory. The algorithm is triggered when the memory size γ defined by the user in XML allocating for each MapTask is parsed. Assuming that the configurable memory of all nodes in the MTB is indiscriminate, if γ obtained from the configuration is greater than λ, the configuration will lose efficacy unquestionably. In such circumstances, the ϵ is adjusted to the maximum configurable memory by default. Conversely, the γ configurated by user is possibly effective when $\gamma \leq \lambda$. Then the algorithm needs to check whether the allocation ϵ can make the available resources fully profitable by the corresponding method (line 4 in Algorithm 2). When the configurable memory is not fully allotted, the strategy (lines 5-7 in Algorithm 2) readjusts ϵ to allocate the largest possible memory for each MapTask while each node obtains the maximum MapTask parallelism.

4 Experiments and Evaluation

4.1 Experimental Design

Under the premise of ensuring the integrity and validity of the transcoded data, we focus on taking the task completion time and speedup as performance evaluation metrics. In experimental design, our main purpose is to evaluate the performance of the proposed system and explore the configuration factors that affect the transcoding efficiency of the system.

More detail, we will first verify the efficiency advantages of distributed transcoding compared to single machine transcoding through experiment in Section 4.3.1, and then compare distributed transcoding and distributed transcoding under different system configurations through Sections 4.3.2,4.3.3,4.3.4. The experiment takes time as an observation variable, and uses the data set size, the number of cluster slave nodes, each stage of the service process, video segmentation size, and task memory allocation [31] as control variables.

Finally, we implemented and extensively evaluated the system CDVT, and obtained design parameters and general recommended transcoding configurations.

4.2 Experimental Setup

We set up a Hadoop cluster consisting of 5 computing nodes to conduct this experiment. One of the computers can serve as the master (NameNode) and the remaining 4 as slaves (DataNode and TaskTracker). Tables 1 gives these software and hardware parameters.

In addition, in order to verify the performance of transcoding source video data of different scales into target data, we give the source and target data parameters used in this experiment in Table 2.

4.3 Experimental Results

In the following experiments, we took time as an observation variable, and uses the data set size, the number of cluster slave nodes, each stage of the service process, video segmentation size, and task memory allocation as control variables.

Table 1: Configuration of Nodes

Components	Configurations and Releases
OS	Ubuntu_14.04_LTS
JDK	Oracle JDK 1.8 Update 171
Hadoop	Apache Hadoop 2.7.6 stable release
FFmpeg	ffmpeg 3.3.8
MKVmerge	mkvmerge v6.7.0 64bit
CPU	Main frequency:3.20GHz, Intel Core i5 4460
RAM	Capacity:4 GB, Kingston DDR3(800MHz)

Table 2: Parameters of Source and Target data

Parameter	Source data	Target data
Codec	Avc1	MPEG-4
Resolution	1920x800	720x480
Container	mkv	avi
Frame rate	30	25
Aspect ratio	16:9	16:9

Table 3: Total transcoding time(s) and speedup(SU) with different cluster size

Nodes	1	2		3		4	
Dataset(GB)	Time(s)	Time(s)	SU	Time(s)	SU	Time(s)	SU
0.25	97.84	72.01	1.36	70.16	1.39	70.90	1.38
0.5	205.36	149.82	1.37	148.74	1.38	148.07	1.39
1	482.72	344.80	1.40	335.22	1.44	312.86	1.54
2	1030.54	682.48	1.51	599.15	1.72	551.09	1.87
4	1920.94	1178.49	1.63	1032.76	1.86	934.86	2.06

4.3.1 Experiment of Cluster Size Affecting Performance

In the first experiment, we took 0.25GB, 0.5GB, 1GB, 2GB, and 4GB video files as inputs, and measured the time-consuming and speedup ratios for different cluster sizes. The speedup is used to describe the increase in the processing rate of distributed transcoding methods relative to single-point transcoding, so it is defined as: Speedup(n) = transcoding time for single-node / transcoding time for n nodes.

Table 3 reveals the transcoding time-consuming and speed-up ratios of different size datasets under different cluster sizes. For example, when the number of slave nodes in the system is 4, it takes 934.86 seconds for 4GB of video data, which has speedup as 2.06 times compared with the completion time of a single machine transcoding of 1920.94 seconds. It also means a reduction in time costs by more than 50%.

Fig. 3, 4 can visually illustrate the effect of various cluster nodes on transcoding performance through the relationship between transcoding time and cluster size. They also reveal as following: (1) when the source video size is constant, the system can use more nodes to obtain less time-consuming and higher speedup; (2) when the cluster size is constant, larger source video can be obtained better transcoding efficiency.

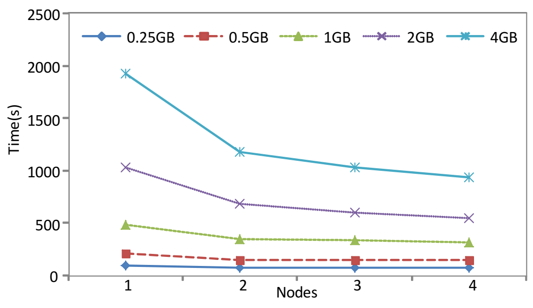

Fig. 3: Transcoding time under different cluster nodes

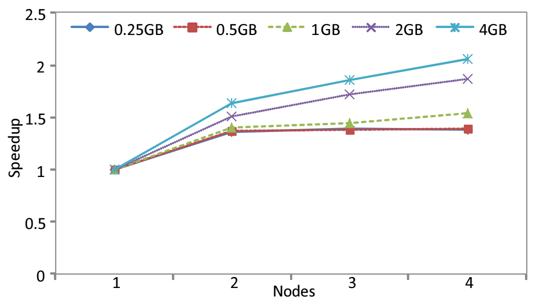

Fig. 4: Speedup under different cluster nodes

Table 4: Time consuming at each stage of the transcoding service

Task	Time(s)				
	0.25GB	0.5GB	1GB	2GB	4GB
Receive and parse XML configuration files	0.51	0.53	0.54	0.50	0.58
Split process of the source data in VDPAMD	0.89	8.97	22.66	41.95	73.47
Create a corresponding index for each split	0.03	0.02	0.03	0.03	0.03
Upload splits and indexes to specific folder in HDFS	11.90	23.93	59.52	121.06	221.66
Transcode on MapReduce-based Transcoding Domain	53.85	105.74	208.87	346.68	565.42
Download and collect all target segments in VDPAMD	3.84	8.55	20.40	39.17	72.13
Merge and get final target data	0.41	0.89	1.39	2.26	4.19
Total	70.90	148.07	312.26	551.09	934.86

4.3.2 Task analysis Experiment in Transcoding Service

In the second experiment, we start to disassemble the entire transcoding service workflow into multiple stages, that is, multiple different tasks, according to the system details described in System Structure Design (Section 3.1) and System Workflow (Section 3.2). We also used 0.25GB, 0.5GB, 1GB, 2GB, and 4GB video files as input to measure the time consumption of tasks in each stage of the transcoding process, and put the experimental data in Table 4.

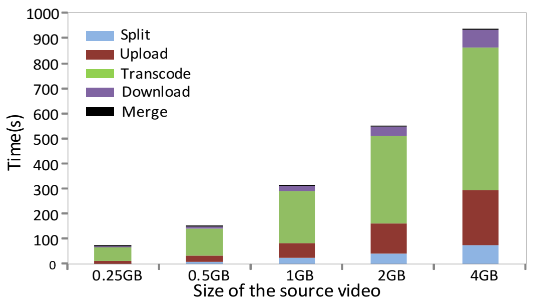

Fig. 5: Time cost for different tasks in transcoding services

For example, for the same task of receiving and parsing XML configuration files, 0.25GB data takes 0.51 seconds, 0.5GB takes 0.53 seconds, 1GB takes 0.54 seconds, 2GB takes 0.50 seconds, and the last 4GB takes 0.58 seconds. This means that throughout the workflow, the time it takes to process the task of receiving and parsing the XML configuration is not much different, and it does not fluctuate significantly due to changes in the size of the original data. Similarly, the task with minimal time fluctuations throughout the workflow also has the task of creating a corresponding index for each split.

In contrast, for the same task of transcoding on the MapReduce-based Transcoding Domain, 0.25GB data takes 53.85 seconds, 0.5GB takes 105.74 seconds, 1GB takes 208.87 seconds, 2GB takes 346.68 seconds, and the last 4GB takes 565.42 seconds. It means that throughout the workflow, the time it takes to process the task of transcoding on the MapReduce-based Transcoding Domain is much different, and it fluctuated significantly due to changes in the size of the original data. Finally, according to Fig. 5, in the input data processing of all sizes, the task of transcoding on the MapReduce-based Transcoding Domain has the largest time-consuming component and the largest fluctuation. Therefore, we can reduce the overall execution time by focusing on further improving the performance of transcoding on the MapReduce-based Transcoding Domain, which is also the most important work in the next section.

4.3.3 Experiment of Split Size Affecting Performance

In the third experiment, we are inspired by the second experiment, which will mainly discuss the factors affecting the performance of transcoding on the MapReduce-based Transcoding Domain. We also took 0.25GB, 0.5GB, 1GB, 2GB, and 4GB video files as input and measured the time-consuming tasks of each stage in the transcoding workflow, but we added a control variable: split size. Split size is the unit of data processing during the video splitting phase, and it will also be the bigness of each task processed by MapTask. Here, we performed five sets of tests on each data set according to the XML file with the split sizes of 64MB, 100MB, 128MB, 150MB, and 256MB, and finally put the obtained data into Table 5.

Table 5: Time cost for different tasks under different split sizes and viarous data sizes

Dataset	Main task	Time(s) cost at different split sizes				
		64MB	100MB	128MB	150MB	256MB
0.25GB	Split	1.42	1.39	0.89	1.12	0.89
	Upload	20.13	16.47	11.90	10.51	9.78
	Transcode	71.16	62.36	53.85	56.71	60.51
	Download	7.87	6.27	3.84	3.57	1.96
	Merge	0.20	0.18	0.41	0.19	0.18
	Total	100.78	86.67	70.90	72.09	73.31
0.5GB	Split	6.38	7.83	8.97	9.25	8.86
	Upload	45.04	28.89	23.93	21.86	19.99
	Transcode	129.34	118.61	105.74	110.62	115.72
	Download	17.97	10.74	8.55	8.25	7.18
	Merge	0.99	0.67	0.88	0.90	0.67
	Total	199.73	166.74	148.07	150.87	152.42
1GB	Split	34.24	26.07	22.67	19.61	21.31
	Upload	98.40	70.03	59.53	52.69	51.10
	Transcode	257.67	227.96	208.87	222.86	239.10
	Download	40.20	25.15	20.40	18.77	11.08
	Merge	1.55	1.42	1.40	1.34	1.55
	Total	432.01	350.63	312.86	315.26	324.13
2GB	Split	43.65	41.85	41.95	45.00	45.79
	Upload	189.46	137.58	121.04	112.06	98.97
	Transcode	475.23	416.13	346.68	384.00	412.45
	Download	74.24	47.76	39.16	33.53	20.24
	Merge	2.58	2.29	2.26	2.25	2.58
	Total	785.16	645.62	551.09	576.85	580.03
4GB	Split	81.35	75.97	73.47	73.27	74.77
	Upload	352.15	267.26	221.66	206.52	188.21
	Transcode	862.39	730.45	565.42	635.30	688.99
	Download	119.91	94.63	72.13	57.87	41.82
	Merge	4.99	4.23	4.19	4.45	4.15
	Total	1420.79	1172.54	934.86	977.40	997.94

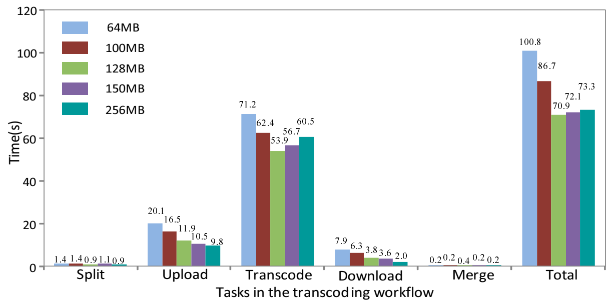

Fig. 6: Time cost for different tasks under different split sizes and 0.25GB data

For instance, for the same stage of transcoding with 4GB dataset (Transcode), it takes 862.39 seconds under split size is 64MB, takes 730.45 seconds under split size is 100MB, takes 565.42 seconds under split size is 128MB, takes 635.30 seconds under split size is 150MB and takes 688.99 seconds under split size is 256MB.

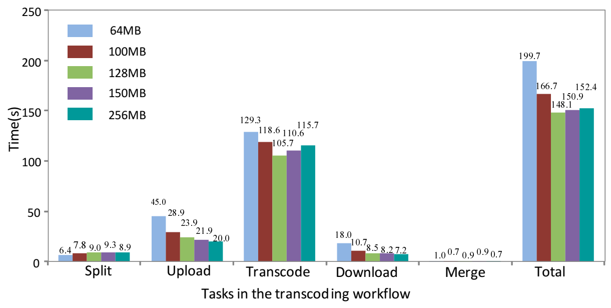
Fig. 7: Time cost for different tasks under different split sizes and 0.5GB data

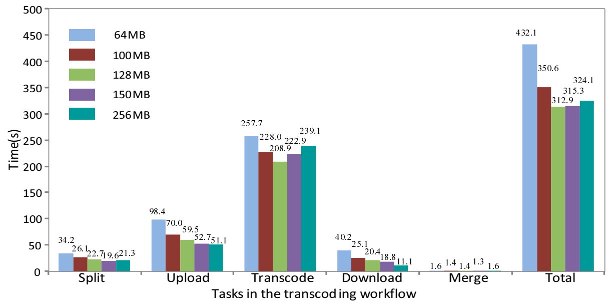
Fig. 8: Time cost for different tasks under different split sizes and 1GB data

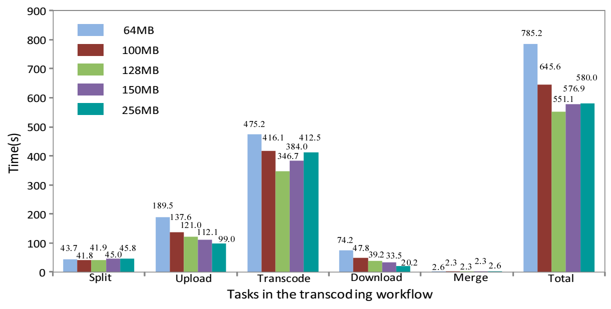
Fig. 9: Time cost for different tasks under different split sizes and 2GB data

Figs. 6-10 more vividly reflect the impact of split size on transcoding performance. Taking Fig. 10 as an example for illustration, we find that as the split size is changed, the time-consuming in the stages of Split and Merge jitter small and this fluctuation has almost no effect on the transcoding time (Total). The Upload and Download tend to decrease as split size increases. It is because the larger the partition size, the smaller the number of data blocks and the smaller the overhead of file transfer. And under various datasets, Fig. 11 shows the global impact of different split sizes on the total transcoding time.

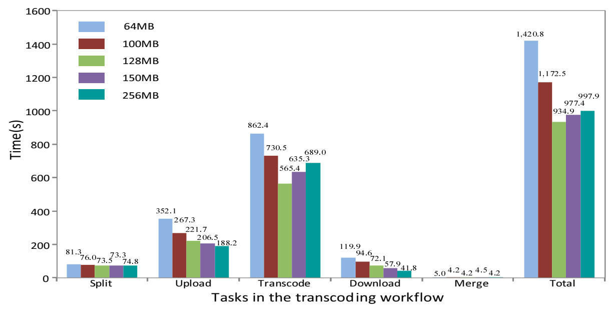

Fig. 10: Time cost for different tasks under different split sizes and 4GB data

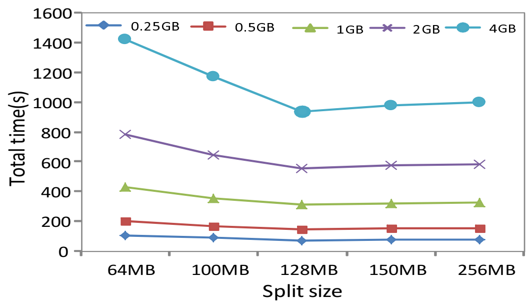

Fig. 11: Transcoding time under different split sizes and various dataset

Most notably, however, as the split size changes, the jittering trend of the transcoding task (Transcode) approaches the convex function and is similar to the trend of total time consumption (Total). It means that the system can improve the performance of the stage of transcoding on the MapReduce-based Transcoding Domain through proper split size configuration, thereby minimizing the total time overhead. The reason why we choose to slice with 128MB of data for the optimized performance may be that the default size of the HDFS data storage unit is 128MB, which can be manually configured. Therefore, we recommend configuring the split size configuration as much as possible to the size of the distributed storage data block in the system design to obtain the optimized system performance.

4.3.4 Experiment of MapTak Allocated Memory Affecting Performance

To further discuss the impact of memory allocation on transcoding performance, the last experiment will be performed with a fixed size of slave nodes (4 nodes) and a splitting strategy of 128 MB. There are also five sizes of data sets to be transcoded.

Table 6: Total transcoding time(s) under various memory allocated to MapTask

Allocated Memory	1GB	2GB	3GB	4GB
Dataset(GB)	Time(s)	Time(s)	Time(s)	Time(s)
0.25	71.88	71.77	72.88	70.90
0.5	170.50	160.89	179.68	148.07
1	366.24	335.22	312.86	312.86
2	631.20	581.36	666.49	551.09
4	1051.20	1006.86	1125.33	934.86

Before discussing the experimental results, we want to introduce what is allocated memory to MapTask. In the MapReduce working framework, there is a parameter named mapreduce.map.memory.mb in its configuration file, which is used to control the processing memory allocated to each MapTask. For example, suppose that on a node with 4GB memory, if memory allocated to MapTask is set to 1GB, then theoretically, at most four tasks can be run simultaneously; if this value is set to 3GB, only one task can be run. Then, the dilemma is that the smaller the allocation of memory, the more parallel the processing of tasks on each node will be, but at the same time the ability to process the same tasks will be weakened. In this case, getting the appropriate compromise is even more important for computationally intensive video transcoding tasks.

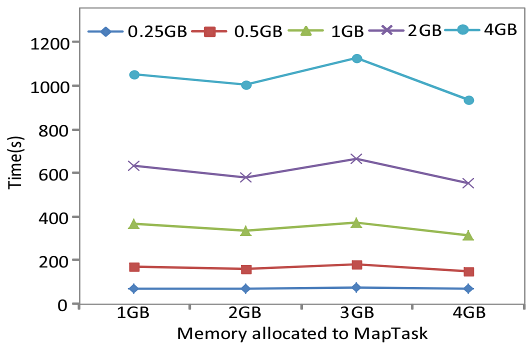

Fig. 12: Time cost under different allocated memory and various dataset

Table 6 and Fig. 12 show the results of transcoding makespan in the final experiment with 0.25GB, 0.5GB, 1GB, 2GB, and 4GB video data as input under different memory allocated to MapTask. It can be found that the transcoding performance of the system is the worst when the allocated memory is 3GB. It is because 3GB is not a factor of the system node memory (4GB), which results in insufficient memory usage. When the memory is allocated to 1GB, 2GB, and 3GB respectively, the system transcoding performance fluctuates small. When the size of the processed data is large, the allocation of memory as large as possible will make the system's transcoding performance the best. It shows that in an effective parallel processing framework, equipping computing nodes with stronger computing capabilities is more beneficial to system transcoding performance.

5 Conclusion and Future Work

In this paper, we proposed the architecture of a distributed video transcoding system that can efficiently transcode non-real-time large-scale video on users' demand, which can be applied to video-on-demand and video sharing over mobile devices. In particular, we expounded the system design and workflow in detail, and discussed the effect of cluster size, intermediate task flow, split strategy, and memory allocation to MapTask on system performance through four sets of experiments. Experimental results show that larger cluster size or befitting memory allocation can produce the optimized system performance and shorter transcoding completion time at the system design parameters level. At the transcoding configuration level, users choose a larger dataset to be processed, or select split size that is closer to the current system storage unit, which can maximize the advantages of the proposed system and will enlighten subsequent applications. In addition, the proposed system can ensure the quality of transcoding tasks while reducing time-to-cost by up to about 50%, when the system is running under optimized configuration parameters.

Future research includes implementing distributed processing of real-time video data or combining transcoding tasks with crowdsourcing under limited computing resources.

Acknowledgement

The research was financially supported by National Natural Science Foundation of China (No. 61972366), the Foundation of Key Laboratory of Network Assessment Technology, Chinese Academy of Sciences (No. KFKT2019-003), Major Scientific and Technological Special Project of Guizhou Province (No. 20183001), and the Foundation of Guizhou Provincial Key Laboratory of Public Big Data (No. 2018BDKFJJ009, No. 2019BDKFJJ003, No. 2019BDKFJJ011).

References

1. S.-I. Hong, H.-S. Lyu, C.-G. In, J.-C. Park, C.-H. Lin, D.-H. Yoon, Development of digital multimedia player based on mobile network sever, in: 2012 14th International Conference on Advanced Communication Technology (ICACT), IEEE, 2012, pp. 1280–1283.
2. S. Petrangeli, N. Bouten, E. Dejonghe, J. Famaey, P. Leroux, F. De Turck, Design and evaluation of a dash-compliant second screen video player for live events in mobile scenarios, in: 2015 IFIP/IEEE International Symposium on Integrated Network Management (IM), IEEE, 2015, pp. 894–897.

3. D. Davis, G. Figueroa, Y.-S. Chen, Socirank: identifying and ranking prevalent news topics using social media factors, IEEE transactions on systems, man, and cybernetics: systems 47 (6) (2016) 979–994.
4. Y. Liu, A. Liu, N. N. Xiong, T. Wang, W. Gui, Content propagation for content-centric networking systems from location-based social networks, IEEE Transactions on Systems, Man, and Cybernetics: Systems 49 (10) (2019) 1946–1960.
5. C. Xu, W. Ren, L. Yu, T. Zhu, K.-K. R. Choo, A hierarchical encryption and key management scheme for layered access control on h. 264/svc bitstream in internet of things, IEEE Internet of Things Journal (2020).
6. A. I. Hentati, L. C. Fourati, O. B. Rhaiem, New hybrid rate adaptation algorithm (hr2a) for video streaming over wlan, in: 2017 Sixth International Conference on Communications and Networking (ComNet), IEEE, 2017, pp. 1–6.
7. Z. Li, Y. Huang, G. Liu, F. Wang, Z.-L. Zhang, Y. Dai, Cloud transcoder: Bridging the format and resolution gap between internet videos and mobile devices, in: Proceedings of the 22nd international workshop on Network and Operating System Support for Digital Audio and Video, ACM, 2012, pp. 33–38. doi:10.1145/2229087.2229097.
8. G. Gao, W. Zhang, Y. Wen, Z. Wang, W. Zhu, Y. P. Tan, Cost optimal video transcoding in media cloud: Insights from user viewing pattern, in: 2014 IEEE International Conference on Multimedia and Expo (ICME), IEEE, 2014, pp. 556–571. doi:10.1109/ICME.2014.6890255.
9. H. Tan, L. Chen, An approach for fast and parallel video processing on apache hadoop clusters, in: 2014 IEEE International Conference on Multimedia and Expo (ICME), IEEE, 2014, pp. 1–6. doi:10.1109/ICME.2014.6890135.
10. H.-W. Kim, H. Mu, J. H. Park, A. K. Sangaiah, Y.-S. Jeong, Video transcoding scheme of multimedia data-hiding for multiform resources based on intra-cloud, Journal of Ambient Intelligence and Humanized Computing (2019) 1–11doi:10.1007/s12652-019-01279-1.
11. FFmpeg, “a complete, cross-platform solution to record, convert and stream audio and video.”, http://ffmpeg.org/ (Apr 2019).
12. Mkvmerge, “mkvtoolnix – matroska tools for linux/unix and windows.”, https://mkvtoolnix.download/doc/mkvmerge.html (Apr 2019).
13. J. Huh, Y.-H. Kim, J. Jeong, Ultra-high resolution video distributed transcoding system using memory-based high-speed data distribution method, in: 2019 34th International Technical Conference on Circuits/Systems, Computers and Communications (ITC-CSCC), IEEE, 2019, pp. 1–4.
14. T.-H. Hsu, Z.-Y. Wang, A distributed shvc video transcoding system, in: 2017 10th International Conference on Ubi-media Computing and Workshops (Ubi-Media), IEEE, 2017, pp. 1–3.
15. X. Li, M. A. Salehi, Y. Joshi, M. K. Darwich, B. Landreneau, M. Bayoumi, Performance analysis and modeling of video transcoding using heterogeneous cloud services, IEEE Transactions on Parallel and Distributed Systems 30 (4) (2018) 910–922.
16. X. Li, M. A. Salehi, M. Bayoumi, N.-F. Tzeng, R. Buyya, Cost-efficient and robust on-demand video transcoding using heterogeneous cloud services, IEEE Transactions on Parallel and Distributed Systems 29 (3) (2017) 556–571. doi:10.1109/TPDS.2017.2766069.
17. S. Sameti, M. Wang, D. Krishnamurthy, Stride: Distributed video transcoding in spark, in: 2018 IEEE 37th International Performance Computing and Communications Conference (IPCCC), IEEE, 2018, pp. 1–8. doi:10.1109/PCCC.2018.8711214.

18. Apache, “the apache hadoop project develops open-source software for reliable, scalable, distributed computing.”, http://hadoop.apache.org/ (Dec 2018).
19. Apache, “hdfs architecture guide.”, http://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html (Dec 2018).
20. Apache, “mapreduce tutorial.”, http://hadoop.apache.org/docs/stable/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html (Dec 2018).
21. F. Yang, Q.-W. Shen, Distributed video transcoding on hadoop, Computer Systems & Applications 11 (2011) 020.
22. C. Ryu, D. Lee, M. Jang, C. Kim, E. Seo, Extensible video processing framework in apache hadoop, in: 2013 IEEE 5th International Conference on Cloud Computing Technology and Science, Vol. 2, IEEE, 2013, pp. 305–310. doi:10.1109/CloudCom.2013.153.
23. V. K. Kodavalla, Transcoding of next generation distributed video codec for mobile video, in: 2018 Second International Conference on Advances in Electronics, Computers and Communications (ICAECC), IEEE, 2018, pp. 1–7.
24. M. Chen, W. Chen, L. Cai, Data-driven parallel video transcoding for content delivery network in the cloud, in: 2018 5th IEEE International Conference on Cyber Security and Cloud Computing (CSCloud)/2018 4th IEEE International Conference on Edge Computing and Scalable Cloud (EdgeCom), IEEE, 2018, pp. 196–199. doi:10.1109/CSCloud/EdgeCom.2018.00042.
25. C. Song, W. Shen, L. Sun, Z. Lei, W. Xu, Distributed video transcoding based on mapreduce, in: 2014 IEEE/ACIS 13th International Conference on Computer and Information Science (ICIS), IEEE, 2014, pp. 309–314. doi:10.1109/ICIS.2014.6912152.
26. A. Heikkinen, J. Sarvanko, M. Rautiainen, M. Ylianttila, Distributed multimedia content analysis with mapreduce, in: 2013 IEEE 24th Annual International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC), IEEE, 2013, pp. 3497–3501.
27. M. Kim, S. Han, Y. Cui, H. Lee, H. Cho, S. Hwang, Clouddmss: robust hadoop-based multimedia streaming service architecture for a cloud computing environment, Cluster Computing 17 (3) (2014) 605–628.
28. M. R. Zakerinasab, M. Wang, Does chunk size matter in distributed video transcoding?, in: 2015 IEEE 23Rd international symposium on quality of service (IWQos), IEEE, 2015, pp. 69–70.
29. M. R. Zakerinasab, M. Wang, Dependency-aware distributed video transcoding in the cloud, in: 2015 IEEE 40th conference on Local computer networks (LCN), IEEE, 2015, pp. 245–252.
30. M. Kim, Y. Cui, S. Han, H. Lee, Towards efficient design and implementation of a hadoop-based distributed video transcoding system in cloud computing environment, International Journal of Multimedia and Ubiquitous Engineering 8 (2) (2013) 213–224. doi:10.1109/icce-tw.2015.7216972.
31. J. Yang, R.-F. Li, A container resource configuration method in hadoop transcoding cluster based on requirements of a sample split, in: 2017 IEEE 2nd International Conference on Cloud Computing and Big Data Analysis (ICCCBDA), IEEE, 2017, pp. 108–112. doi:10.1109/ICCCBDA.2017.7951893.